06

Retornando excepciones

Transcripción

[00:00] Ya hemos recordado muy bien cómo es una excepción checked y una unchecked, ya hemos visto los diversos tratamientos que les podemos dar, las alternativas que tenemos y en qué casos necesitamos hacer un tratamiento del excepción obligatoriamente y en qué casos no.

[00:17] Ahora para afianzar más este concepto, vamos sí a ver nuestro proyecto de herencia y polimorfismo que hicimos en el curso anterior, que es nuestro Byte Bank, el proyecto que hemos venido avanzando desde el primer módulo de su formación en Java.

[00:37] Entonces, bueno, yo voy a guardar aquí y voy a cerrar todas estas clases para tener limpio mi espacio de trabajo. Y aquí yo voy a abrir la clase cuenta. Entonces vamos a recordar un poco cómo era esta estructura. Yo tengo una clase abstracta cuenta de la cual extienden dos clases hijas que son estas dos que están aquí: CuentaAhorros, CuentaCorriente, cada una con su propia lógica de método deposita.

[01:11] Recordemos que las clases hijas pueden sobrescribir los métodos y añadir su propia lógica. Entonces, ahora aquí yo tengo un constructor por defecto, tengo un constructor en el cual yo envío el número de agencia, número de cuenta obligatoriamente, el método deposita es del tipo abstracto, tengo el método saca. ¿Entonces, qué es lo que vamos a hacer aquí?

[01:42] Recordemos que en los primeros ejemplos que vimos para el tratamiento de excepciones yo mencioné mucho el método saca. ¿Por qué?

Porque aquí la única condición que yo estoy escribiendo para que él no pueda sacar dinero, es que simplemente no tenga saldo, pero ya vimos que pueden haber muchos más motivos por los cuales yo no pueda retirar dinero.

[02:05] Ya sea que mi cuenta está bloqueada, que el banco detectó un intento de fraude que no tengo saldo y entre otros, entonces yo voy a implementar aquí esa lógica, usando excepciones. ¿Cómo lo voy a hacer? Primero que todo tengo que definir el tipo de método que yo voy a usar aquí. ¿Por qué? Porque saca era un booleano que me decía si puedo o no puedo sacar.

[02:32] Pero también podría ser que simplemente saca sea un void que si saco, perfecto, saqué mi dinero, y si no saco, él me va a decir por qué no saqué con una excepción, con un error. Para eso, primero voy a crear mi excepción. Entonces voy aquí, Byte Bank, new class. Y como va a ser una excepción, voy a llamarla SaldoInsuficienteException, y recordemos que yo no necesito agregar ese exception al final.

[03:18] Yo no estoy obligado a que MiException tenga ese post de Exception, pero es buena práctica y es ya una convención que si va a ser una Exception yo le diga, pues en el nombre de la clase que en efecto va a ser un Exception. Entonces, le doy aquí finish, y en SaldoInsuficienteException yo voy a decirle, pues que va a ser un RuntimeException para no hacer mucho problema con la compilación del código que usa eso.

[03:46] Entonces yo le voy a decir aquí extends RuntimeException, perfecto. ¿Y aquí qué puedo hacer yo? Puedo llamar a los constructores de la clase padre de RuntimeException, y para este caso yo voy a llamar solamente al constructor que usa un mensaje, que envía un mensaje, entonces yo voy a String mensaje, abro aquí y llamo a super, entre paréntesis, punto y coma, y que le envió como parámetro el mensaje. Y listo.

[04:31] ¿Qué estoy diciendo con esto? Declaro el constructor de SaldoInsuficienteException y cada vez que yo lance esa excepción, yo voy a

mandarle un mensaje y él mediante el constructor de la clase padre que es RuntimeExceptin, él va a setear ese mensaje. Hasta ahí todo bien.

[04:48] No hay más más ciencia aquí hasta el momento. Volvemos a cuenta y vamos a nuestro método saca. Nuestra primera condición aquí es de que si mi saldo es mayor al valor, entonces ahí yo puedo sacar dinero, le resto el valor al saldo, perfecto y retorno true. Yo voy a sacar esta condición de aquí. Ya van a ver por qué.

[05:19] Bueno, primero que todo yo voy a borrar todo esto de aquí, el else, el return True, porque ya no voy a retornar un booleano. Y esto de aquí lo voy a volver void. Perfecto. ¿Con esto qué estoy diciendo? Con esto lo que yo digo es bueno, si es que sacas dinero, perfecto, se va a restar el valor que quieres sacar de tu saldo. Y si no, pues, te voy a tener que devolver una exception.

[05:52] ¿Y para esto, yo que voy a decir aquí? Le voy a preguntar una cosa. ¿En qué caso yo voy a lanzar esa exception de saldo suficiente? Si this.saldo es menor a valor. Si es igual, entonces perfecto, puedes sacar, te quedas en cero, pero si fuera menor, entonces tú no puedes sacar. Y dénse cuenta que yo cambié la condición del if anterior.

[06:17] Si recordamos un poco, simplemente aquí lo que preguntaba era si yo tenía igual o mayor saldo para retirar, pero aquí yo le estoy preguntando si él tiene menos saldo que lo que él quiere retirar. ¿Por qué? Porque si esta condición es verdadera, ¿entonces yo qué voy a hacer aquí? Voy a lanzar un nuevo throw new SaldoInsuficienteException.

[06:41] ¿Con qué mensaje? Perdí el cursor aquí. "No tienes saldo", y al final su punto y coma. Perfecto. Entonces, es de esta forma que yo consigo tener un control de errores a través de las excepciones. ¿Por qué lo puse aquí arriba y no aquí abajo? Porque generalmente una buena práctica es todo lo que sea control de errores, validaciones, va al inicio del método.

[07:15] Entonces, antes de yo efectuar cualquier operación, yo tengo que validar que los parámetros que yo tengo actualmente sean válidos. Tengo que verificar en este caso que tengo el saldo suficiente para retirar el dinero que yo estoy especificando en valor. Y bueno, con esto tendríamos, pues ya terminado terminado este refactor. Refactor es la reconstrucción del código para mejorarlo.

[07:43] Ahora ya no devuelvo un booleano, sino yo uso simplemente un void. ¿Y a quién transfiere? Bueno, él está aplicando aquí la condición de que si el saldo es mayor, entonces saca de un lado, deposita en la otra cuenta, perfecto, no vamos a modificar el método transfiere, hemos modificado solo el método saca. Y vemos que aquí una de las cuentas dejó de compilar y es la clase de cuenta corriente.

[08:13] ¿Por qué? Porque él está haciendo un overwrite, él está sobrescribiendo un método del tipo boolean que se llama saca, pero en la clase padre, que es la clase cuenta, ya no tenemos un método del tipo boolean con el nombre saca. Tenemos un método del tipo void.

[08:33] Recordando un poco sobre herencia y polimorfismo, en esta clase de aquí, si yo quito el overwrite, si yo comento esta línea de aquí, entonces este código de cualquier forma tampoco va a compilar porque él está de alguna forma llamando al método de la clase padre. Entonces, yo voy a dejar aquí el overwrite con el cual yo especifico que sí en efecto yo quiero sobrescribir el método, quiero que el compilador se asegure de eso.

[09:15] Y principio básico, si yo sobrescribo el método de la clase padre, la firma del método tiene que ser exactamente la misma, por lo cual, si aquí es un método del tipo void, aquí él también tiene que ser del tipo void, primer punto. Guardamos aquí y vemos que en esta línea él ya compila. Me dice: "Perfecto. Tú estás sobrescribiendo el mismo método". ¿Pero ahora cuál es el problema?

[09:38] Que yo acá estoy retornando el método saca de la clase padre y el método del tipo void no debería retornar ningún valor, entonces yo podría dejarlo simplemente así de esta forma y listo, el método ya va a compilar. ¿Por qué? Porque él va a llamar al método de la clase padre y en la clase cuenta él va a efectuar esta operación, y si es que no hay saldo, pues entonces no hay saldo y va a darnos la excepción.

[10:09] Perfecto. Entonces vamos a hacer un testo de la cuenta. Vamos a venir aquí, vamos a crear un nuevo test para no interferir con lo que ya hemos trabajado hasta ahora en ese proyecto. Recordemos que es un proyecto de un módulo anterior del curso. Aquí le vamos a poner TestCuentaExceptionSaldo. Finish

[10:40] Y como ya es de costumbre, vamos a crear nuestro método main, vamos a crear nuestra cuenta = new, y va a ser del tipo CuentaAhorros, con un número de agencia vamos a inventar 123 y un número de cuenta 456. Perfecto. Entonces hemos creado aquí ya nuestra cuenta de ahorros. Nuestra cuenta de ahorros tiene sobrescrito el método deposita pero no el método saca, primer punto que vamos a hacer aquí.

[11:21] Entonces, ¿qué vamos a hacer aquí? Depositar dinero. Cuenta.deposita, vamos a depositarle 200, de cualquier moneda que ustedes quieran. Y ahora vamos a hacer un cuenta.saca y vamos a sacar pues 200. Y vamos a ver qué es lo que sucede aquí. Guardamos aquí, perfecto. Tengo una cuenta 456 con nombre cuenta, perfecto, creó la cuenta, depositó y sacó.

[11:58] Y si yo hiciera esto de aquí, SaldoInsuficienteException. ¿Por qué? Porque yo le dije: "Deposita 200, saca 210" y la operación no se pudo completar, y aquí está el detalle: SaldoInsuficienteException, y el mensaje de ahí "no tiene saldo". Entonces es de esta forma que yo consigo tener un control de errores mucho más detallado usando las excepciones.

[12:29] Si yo le dijera por ejemplo, le doy igual saldo para que retire todo y deje la cuenta en cero, perfecto, perfecto, él va a crear la cuenta y mi saldo va a estar en cero. Si yo aquí le vuelvo a dar cuenta.saca y 10 más, entonces voy a ejecutar, va a la cuenta y nuevamente no tiene saldo. ¿Por qué? Porque yo aquí ya saqué todo.

[12:56] De esta forma pueden ir jugando, yo aquí hice saca dos veces, una de 200 y una de 10, y es exactamente el valor que yo deposité. Podemos ir jugando para ver cómo es que funciona esto de las excepciones. Ya vimos aquí que en la clase cuenta yo podría tener n condiciones para retornar n excepciones, relacionadas con las reglas del negocio.

[13:29] En este caso, ¿cuál es la regla del negocio que yo quise validar? Que para retirar dinero necesitas tener saldo suficiente. Pueden haber n reglas del negocio y las excepciones no sirven para validar que esas reglas sean cumplidas.

[13:41] Entonces especificamos el mensaje de la excepción, tenemos la excepción. Ya sabemos pues cómo implementar este concepto de excepciones y crear nuestros propios tipos de excepciones al mundo real, en este caso en